1. INTRODUCTION

Raveen de Silva, r.desilva@unsw.edu.au
office: K17 202
Course Admin: Anahita Namvar, cs3121@cse.unsw.edu.au

School of Computer Science and Engineering
UNSW Sydney

Term 2, 2022

Table of Contents

1. Admin
2. Solving problems using algorithms
3. Proofs
4. An example of the role of proofs
5. Puzzles

Required knowledge and skills

- Understanding of fundamental data structures and algorithms
  - Arrays, trees, heaps, sorting, searching, etc.
- Written communication skills
  - No programming involved - see COMP4121 and COMP4128
  - Smarthinking for writing help

Prerequisite courses

- Only prerequisite: COMP2521/9024
- Desirable (but not officially required)
  - For undergrads:
    - MATH1081 Discrete Mathematics (proofs, graphs)
    - MATH1131/1141 (matrices, complex numbers, limits)
  - For postgrads:
    - COMP9020 Foundations of Computer Science

Extended courses

- The extended courses COMP3821/9801 run in T1 only
- Differences in content and assessment
- Marks will be adjusted in both courses so as not to disadvantage the extended students

Classes

- Lectures (weeks 1–10)
  - Thursday 16:00 - 18:00
  - Friday 11:00 - 13:00
  - At least one revision lecture in week 6 (flexibility week)
  - Live streams and recordings on YouTube
  - Slides on Moodle

- Consultation (weeks 1–10)
  - Tuesday 14:00 - 15:00
  - Friday 14:00 - 15:00
  - Join live on Zoom, recordings on YouTube
  - Exam consultation TBA

Getting Help

- Join the Ed forum!
- No conventional tutorials or labs.
- Help sessions
  - Every weekday from week 2 at 11am and 5pm, except Thursday PM and Friday AM
  - Tutor-led discussion of tutorial problems
  - Voluntary participation
  - Some face-to-face, others on Zoom (recorded)

Assessment

- Assignments
  - Four assignments, released approx bi-weekly
  - Each consists of 4 questions
  - Each weighted 15% of course mark
- Final Exam
  - 8 MCQ, 4 extended response
  - INSPERA
  - Weighted 40% of course mark
- Forum participation
  - Up to 5 bonus marks

Textbooks

Recommended textbook

Kleinberg and Tardos: *Algorithm Design*
paperback edition available at UNSW Bookshop

- excellent: very readable textbook (and very pleasant to read!);
- not so good: as a reference manual for later use.

An alternative textbook

Cormen, Leiserson, Rivest and Stein: *Introduction to Algorithms*
3rd edition also available at UNSW Bookshop, 4th edition not yet

- excellent: to be used later as a reference manual;
- not so good: somewhat formalistic and written in a rather dry style.

Feedback

- Changes from last term:
  - tutor-led help sessions, recordings and F2F
  - changes to tutorial exercises and solutions
  - assignment question format
- Feedback is always welcome, e.g.
  - myExperience survey
  - feedback post on Ed (can post anonymously)
  - email

Table of Contents

1. Admin

2. Solving problems using algorithms

3. Proofs

4. An example of the role of proofs

5. Puzzles

Introduction

What is this course about?

It is about **designing algorithms** for solving practical problems.

What is an algorithm?

- An algorithm is a collection of precisely defined steps that are executable using certain specified mechanical methods.
- By "mechanical" we mean the methods that do not involve any creativity, intuition or even intelligence. Thus, algorithms are specified by detailed, easily repeatable "recipes".
- The word "algorithm" comes by corruption of the name of Muhammad ibn Musa al-Khwarizmi, a Persian scientist 780–850, who wrote an important book on algebra, *"Al-kitab al-mukhtasar fi hisab al-gabr wa'l-muqabala"*.

Introduction

In this course we will deal only with sequential deterministic algorithms, which means that:

- they are given as sequences of steps, thus assuming that only one step can be executed at a time;
- the action of each step gives the same result whenever this step is executed for the same input.

Introduction

Why should you study algorithm design?

Can you find every algorithm you might need using Google?

Our goal:

To learn **techniques** which can be used to solve **new, unfamiliar** problems that arise in a rapidly changing field.

Course content:

- a survey of algorithm **design techniques**
- particular algorithms will be mostly used to illustrate design techniques
- emphasis on development of your algorithm design **skills**

Example: Two Thieves

Problem

Alice and Bob have robbed a warehouse and have to split a pile of items without price tags on them. Design an algorithm to split the pile so that each thief **believes** that they have got at least half the loot.

Solution

Alice splits the pile in two parts, so that she believes that both parts are of equal value. Bob then chooses the part that he believes is no worse than the other.

Example: Two Thieves

Note

We are assuming that it's always possible to split up the loot into whatever fraction we like. With discrete items this is more complicated than it might appear!

Question

If there are n items, and Alice values the ith at v_i dollars, can Alice efficiently split the loot into two equal piles?

Answer

There is no known algorithm that is significantly more efficient than the brute force (try all choices, of which there are approx 2^n).

Example: Three Thieves

Problem

Alice, Bob and Carol have robbed a warehouse and have to split a pile of items without price tags on them. How do they do this in a way that ensures that each thief believes that they have got at least one third of the loot?

Example: Three Thieves

- The problem is much harder with 3 thieves!
- Let us try do the same trick as in the case of two thieves. Say Alice splits the loot into three piles which she thinks are of equal value; then Bob and Carol each choose which pile they want to take.
- If they choose different piles, they can each take the piles they have chosen and Alice gets the remaining pile; in this case clearly each thief thinks that they got at least one third of the loot.

Example: Three Thieves

- But what if Bob and Carol choose the same pile?
- One might think that in this case, Alice can pick either of the other two piles, after which the remaining two piles are put together for Bob and Carol to split them as in the earlier problem with only two thieves.
- Unfortunately this does not work!

Example: Three Thieves

- Suppose that Alice splits the loot into three piles X, Y, Z, and that Bob thinks that

$$X = 50\%, Y = 40\%, Z = 10\%$$

of the total value, while Carol thinks that

$$X = 50\%, Y = 10\%, Z = 40\%.$$

Example: Three Thieves

- Clearly both Bob and Carol choose pile X, so Alice can choose pile Y or Z.
- However, if Alice picks pile Y, then Bob will object that (in his eyes) only 60% of the loot remains, so he is not guaranteed to get at least one-third of the total.
- If instead Alice picks pile Z, then Carol will object for the same reason.
- What would be a correct algorithm?

Example: Three Thieves

Algorithm

- Alice makes a pile X which she believes is $1/3$ of the whole loot.
- Alice proceeds to ask Bob whether he agrees that $X \leq 1/3$.
- If Bob says YES, then he would be happy to split the remainder of the loot (worth $\geq 2/3$) with one other thief.
    - Alice then asks Carol whether she thinks that $X \leq 1/3$.
    - If Carol says NO, then Carol takes X, and Alice and Bob split the rest.
    - If Carol says YES, then Alice takes X, and Bob and Carol split the rest.

Example: Three Thieves

Algorithm (continued)

- What if Bob says NO? Then Alice values pile X at $1/3$ of the total, but Bob believes it to be $> 1/3$.
- Now we ask Bob to reduce the pile X until he believes it to be $1/3$ of the total. Alice values the new pile as $< 1/3$, so she is happy to split the remainder of the loot (worth $> 2/3$) with one other thief.
- This is exactly the situation we had before, but with Alice and Bob's roles reversed!
    - Bob asks Carol whether she thinks that $X \leq 1/3$.
    - If Carol says NO, then Carol takes X, and Alice and Bob split the rest.
    - If Carol says YES, then Bob takes X, and Alice and Carol split the rest.

Example: n Thieves

Exercise

Try generalising this to n thieves.

Hint

There is a *nested recursion* happening even with 3 thieves!

Table of Contents

1. Admin
2. Solving problems using algorithms
3. Proofs
4. An example of the role of proofs
5. Puzzles

The role of proofs in algorithm design

Question

When do we need to give a **mathematical proof** that an algorithm we have just designed terminates and returns a solution to the problem at hand?

Answer

When this is not obvious by inspecting the algorithm using common sense!

Mathematical proofs are **NOT** academic embellishments; we use them to justify things which are not obvious to common sense!

Example: MERGE-SORT

Algorithm

```
MERGE-SORT($A, \ell, r$)                    *sorting $A[\ell..r]$*
  1. if $\ell < r$
  2.     then $m \leftarrow \lfloor \frac{\ell+r}{2} \rfloor$
  3.           MERGE-SORT($A, \ell, m$)
  4.           MERGE-SORT($A, m+1, r$)
  5.           MERGE($A, \ell, m, r$)
```

Example: Merge-Sort

- The depth of recursion in Merge-Sort is $\log_2 n$.
- On each level of recursion, merging all the intermediate arrays takes $O(n)$ steps in total.
- Thus, Merge-Sort always terminates, and in fact it terminates in $O(n \log_2 n)$ steps.
- Merging two sorted arrays always produces a sorted array, thus, the output of Merge-Sort will be a sorted array.
- The above is essentially a proof by induction, but we will never bother formalising proofs of (essentially) obvious facts.

The role of proofs in algorithm design

- However, sometimes it is **NOT** clear from a description of an algorithm that such an algorithm will not enter an infinite loop and fail to terminate.
- Sometimes it is **NOT** clear that an algorithm will not run in exponentially many steps (in the size of the input), which is usually almost as bad as never terminating.
- Sometimes it is **NOT** clear from a description of an algorithm why such an algorithm, after it terminates, produces a desired solution.

The role of proofs in algorithm design

- Proofs are needed for such circumstances; in a lot of cases they are **the only way** to know that the algorithm does the job.
- For that reason we will **NEVER** prove the obvious (the CLRS textbook sometimes does just that, by sometimes formulating and proving trivial little lemmas, being too pedantic!). We will prove only what is genuinely nontrivial.
- **However, BE VERY CAREFUL what you call trivial!!**

Table of Contents

1. Admin
2. Solving problems using algorithms
3. Proofs
4. An example of the role of proofs
5. Puzzles

Stable Matching Problem

- Suppose there are n hospitals in the state, and n new doctors have graduated from university. Each hospital wants to hire exactly one new doctor.
- Every hospital submits a list of preferences, which ranks all the doctors, **and** every doctor submits a list of preferences, which ranks all the hospitals.
- Design an algorithm which produces a *stable matching*, which is: a set of n pairs $p = (h, d)$ of a hospital h and a doctor d so that the following situation never happens:

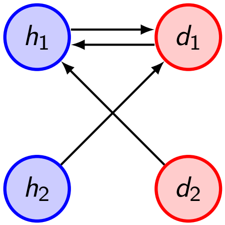
  for two pairs $p = (h, d)$ and $p' = (h', d')$:
  - hospital h prefers doctor d' to doctor d, **and**
  - doctor d' prefers hospital h to hospital h'.

Stable Matching Problem: Example 1

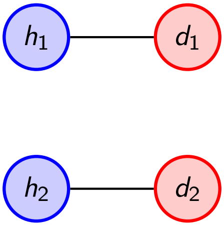
$h_1 : d_1, d_2$ $\qquad$ $d_1 : h_1, h_2$

$h_2 : d_1, d_2$ $\qquad$ $d_2 : h_1, h_2$

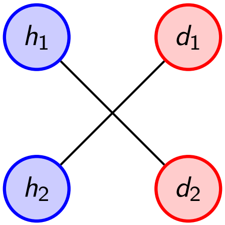
Preferences



Stable



Not stable

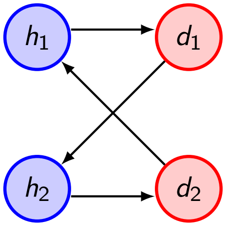


Stable Matching Problem: Example 2

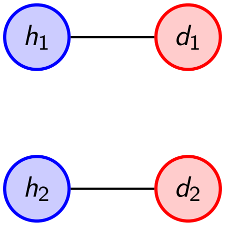
$h_1 : d_1, d_2$ $\qquad$ $d_1 : h_2, h_1$

$h_2 : d_2, d_1$ $\qquad$ $d_2 : h_1, h_2$

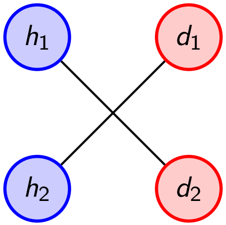
Preferences



Stable



Stable



Stable Matching Problem: $n = 2$

Exercise

Up to symmetry, there are two more cases to consider. Solve them.

The following rules cover all situations with two hospitals and two doctors.

- If the hospitals prefer different doctors, assign each hospital their preferred doctor. Neither hospital wants to swap, so the matching is stable.
  - The same applies vice versa, i.e. if the doctors prefer different hospitals.
- However, if both hospitals prefer the same doctor d **and** both doctors prefer the same hospital h, pair h with d and the other hospital with the other doctor. Neither h nor d wants to swap, so the matching is stable.

Stable Matching Problem: Gale - Shapley algorithm

Question

Given n hospitals and n doctors, how many ways are there to match them, without regard for preferences?

Answer

$n! \approx (n/e)^n$ - more than exponentially many in n ($e \approx 2.71$).

Question

Is it true that for every possible collection of n lists of preferences provided by all hospitals, and n lists of preferences provided by all doctors, a stable matching always exists?

Answer

YES, but this is **NOT** obvious!

Stable Matching Problem: Gale - Shapley algorithm

Question

Can we find a stable matching in a reasonable amount of time?

Answer

YES, using the **Gale - Shapley algorithm**.

- Produces pairs in stages, with possible revisions
- A hospital which has not been paired with a doctor will be called *free*.
- Hospitals will be offering jobs to doctors. Doctors will decide whether they accept a job offer or not.
- Start with all hospitals free.

Stable Matching Problem: Gale - Shapley algorithm

```
While there exists a free hospital which has not offered jobs to all
      doctors, pick such a free hospital $h$ and have it offer a job
      to the highest ranking doctor $d$ on its list to whom it has
      not offered a job yet;
      If    no one has offered $d$ a job yet
            they always accept and a pair $p = (h, d)$ is formed;
      Else  they are already in a pair $p' = (h', d)$;
            If    $h$ is higher on her preference list than $h'$:
                  the pair $p' = (h', d)$ is deleted,
                  $h'$ becomes a free hospital, and
                  a new pair $p = (h, d)$ is formed;
            Else  $h$ is lower on their preference list than $h'$:
                  the job offer is rejected and $h$ remains free.
```

Stable Matching Problem: Gale - Shapley algorithm

Claim 1

The algorithm terminates after $\leq n^2$ rounds of the *While* loop.

Proof

- In every round of the *While* loop one hospital offers a job to one doctor.
- Every hospital can make an offer to a doctor at most once.
- Thus, every hospital can make at most n offers.
- There are n hospitals, so in total they can make $\leq n^2$ offers.
- Thus the *While* loop can be executed no more than n^2 many times.

Stable Matching Problem: Gale - Shapley algorithm

Claim 2

The algorithm produces a matching, i.e., every hospital is eventually paired with a doctor (and thus also every doctor is paired to a hospital).

Proof

- Assume that the *While* loop has terminated, but hospital h is still free.
- This means that h has already offered a job to every doctor.
- Thus, every doctor is paired with a hospital, because a doctor is not paired only if no hospital has offered them a job.
- But this would mean that n doctors are paired with all of n hospitals so h cannot be free. **Contradiction!**

Stable Matching Problem: Gale - Shapley algorithm

Claim 3

The matching produced by the algorithm is stable.

Proof

Note that during the *While* loop:

- a doctor is paired with hospitals of increasing ranks on their list, and
- a hospital is paired with doctors of decreasing ranks on its list.

Assume now the opposite, i.e. that the matching is not stable. Thus, there are two pairs $p = (h, d)$ and $p' = (h', d')$ such that:

$$h \text{ prefers } d' \text{ over } d;$$
$$d' \text{ prefers } h \text{ over } h'.$$

Stable Matching Problem: Gale - Shapley algorithm

Proof (continued)

- Since h prefers d' over d, it must have made an offer to d' before offering the job to d.
- Since h is paired with d, doctor d' must have either:
  - rejected h because they were already at a hospital they prefer to h, or
  - accepted h only to later rescind this and accept an offer from a hospital they prefer to h.
- In both cases d' would now be at a hospital which they prefer over h. **Contradiction!**

Table of Contents

1. Admin
2. Solving problems using algorithms
3. Proofs
4. An example of the role of proofs
5. Puzzles

Puzzle

Why puzzles? It is a fun way to practice problem solving!

Problem

Tom and his wife Mary went to a party where nine more couples were present.

- Not every one knew everyone else, so people who did not know each other introduced themselves and shook hands.
- People who knew each other from before did not shake hands.
- Later that evening Tom got bored, so he walked around and asked all other guests (including his wife) how many hands they had shaken that evening, and got 19 different answers.
- How many hands did Mary shake?
- How many hands did Tom shake?

That's All, Folks!!